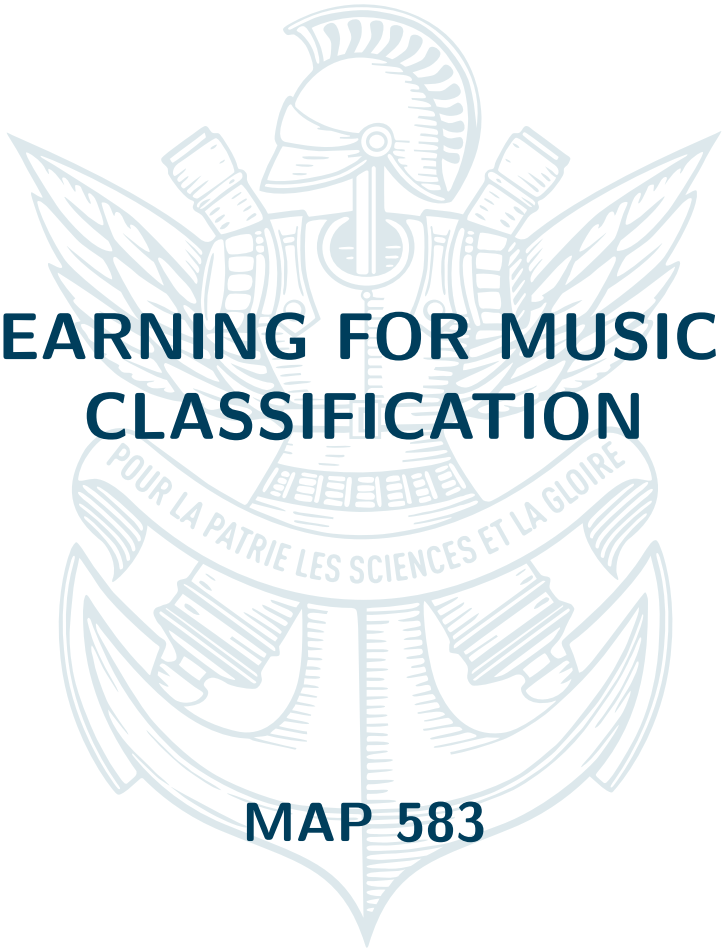


DEEP LEARNING FOR MUSIC GENRE CLASSIFICATION

MAP 583

2018

Barthélemy Duthoit
Antoine Hoorelbeke

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

The recent improvements in deep learning due to the progress in the theoritical background and also to the bigger computational power available brings more and more applications.

Music is omniprsent on Internet, and one of the most important infomation about a song is its music genre. As of today, genres are most often labelled by humans, but one can easily imagine a deep learning framework which can automate this tedious task.

In this report, we will expose the different models and features we elaborated to predict the music genre of a song, using the Magnatagatune data set, which is composed of 30 seconds music extracts of a dozen of music genres.

2
THE DATASET

2.1 About the Dataset

It is hard to come by a dataset that is exploitable for obvious copyright reasons. We wanted to build a neural network model using raw data (mp3). While some datasets offer the possibility of dealing with data that has been pre treated, only a couple have a sufficient amount of raw mp3 to train a neural network. This led us to choose the Magnatagatune dataset that contains 25,863 music extracts of 30 seconds, which can be found here [1].

The only last issue to solve before starting our exploring the data was that the data is unlabelled. On the page of the dataset we were able to find a csv that lists the songs and their genre. However the strings used the names of the songs and that on the mp3s were not the same. Therefore we had to write a script to map the raw mp3 files to their genre. However by doing so we lost about half the data (songs that we were not able to map) which reduced the size of our dataset to 10,747 songs and 10 music genres :

— Alt Rock
— Ambient
— Classical
— Electro Rock
— Electronica
— Hard Rock
— Hip-Hop
— Jazz
— New Age
— World

2.2 Data exploration

2.2.1 • Dimensionality Reduction

We performed dimensionality reduction on our data set in order to see if it was coherent. We performed the dimensionality reduction algortihms on the mel-spectrograms of the songs. We'll explain in a next section what is a mel spectrogram and why we used them.

After trying PCA, LDA and UMAP, LDA is cleary performing better than other dimensionality reduction algorithms.

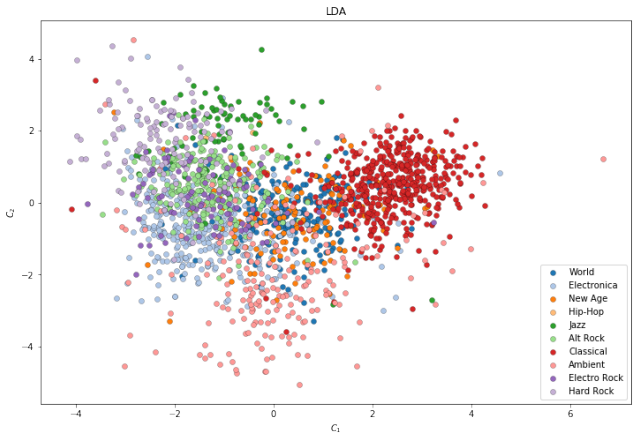


Figure 1 – 2D LDA

The clusters are not clearly distincts, but we can see some paterns. These are even more obvious when reducing the problem to 3 dimensions (Classical music clearly stands out, for it uses very distinct sonorities).

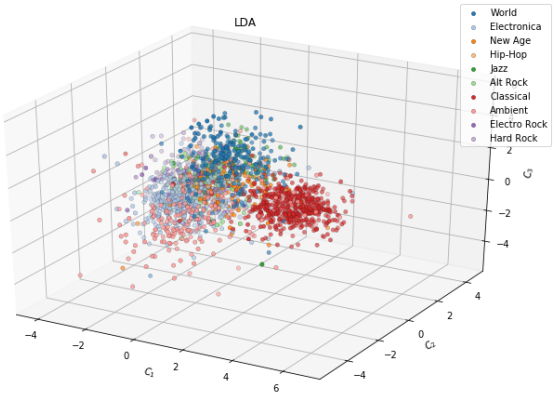


Figure 2 – 3D LDA

The data looks rather coherent, and it doesn't seems like there is any major outlier in the dataset.

2.2.2 • Data Distribution

The classes and not extremely well balanced (Hip-Hop is clearly lacking instances, whilst Classical music and Electronica are over-represented), which might be an issue when training the model.

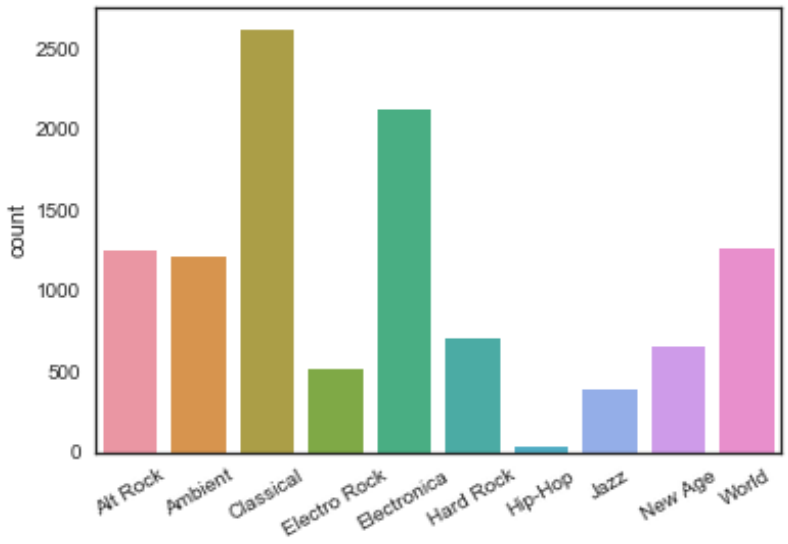


Figure 3 – Data Distribution by Genre

3 DATA ENGINEERING

We decided to exploit the audio features in a completely uninstinctive way. Indeed, instead of directly feeding the neural network with the audio features, we chose to convert the audio features into spectograms.

3.1 Regular Spectograms

The spectrogram of a sound can be seen as an image, thus we thought that using convolutional neural networks (CNN) would be a good idea. Indeed, CNN are the state of art technique in matter of picture recognition. We can see that in regards of the performance of different algorithms to recognize the digits of the MNIST dataset (see [2]), or to classify cats and dogs pictures.

To plot the spectograms, we used python's scipy library and ran the following code to generate the spectograms (after having converter the MP3s to WAVs through Python) :

```
import scipy.io.wavfile
import os
import numpy as np
import matplotlib.pyplot as plt

file = "wav/Alt Rock/burnshee_thornside-rock_this_moon-07-bang_i_shot_him
    -175-204.wav"

rate,audData=scipy.io.wavfile.read(file)
channel1=audData #left
fig,ax = plt.subplots(1)
fig.subplots_adjust(left=0,right=1,bottom=0,top=1)
Pxx, freqs, bins, im = plt.specgram(channel1, Fs=rate, NFFT=1024, cmap="autumn")
```

3.2 Introducing mel-spectograms

While doing research on previous work that had previously been done on the subject [3] [4] [5], we noticed that other projects used mel-spectograms. Those represent the frequencies on a mel-scale :

$$f_{mel} = 1127 \times ln(1 + \frac{f}{700}) \tag{1}$$

Using mel-spectograms better represents the way we, humans, naturally hear sounds. To compute those spectogram we used librosoa :

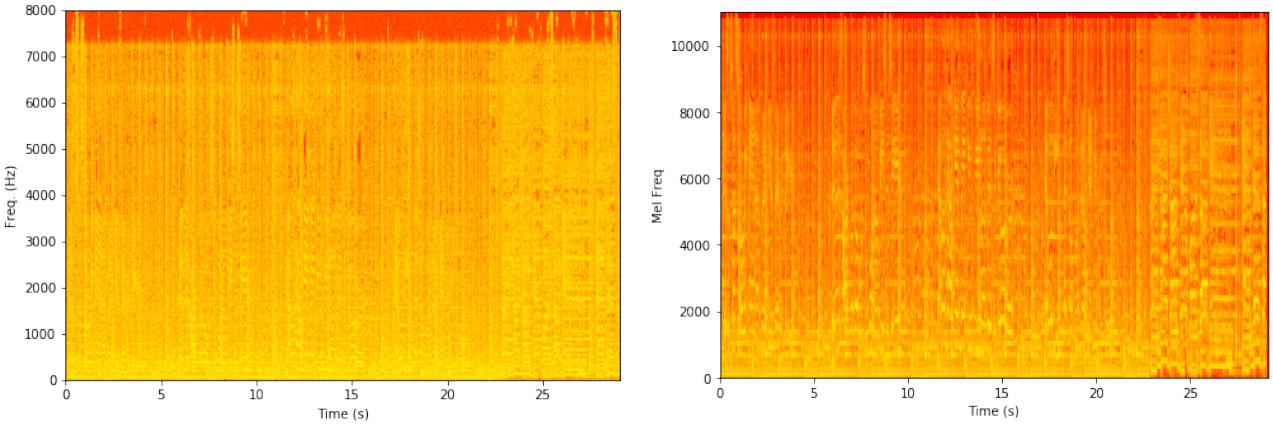
```
import os
import matplotlib.pyplot as plt
import librosa
import librosa.display
import numpy as np

file = "wav/Alt Rock/burnshee_thornside-rock_this_moon-07-bang_i_shot_him
    -175-204.wav"

sig, fs = librosa.load(file)
fig,ax = plt.subplots(1)
fig.subplots_adjust(left=0,right=1,bottom=0,top=1)
S = librosa.feature.melspectrogram(y=sig, sr=fs, fmax=8000)
librosa.display.specshow(librosa.power_to_db(S, ref=np.max), y_axis='linear',
    x_axis='time', cmap="autumn")
```

3.3 Comparaison

Let's compare the two spectograms outputed by the scripts :



(a) Regular spectogram

(b) Mel-spectogram

Figure 4 – Burnshee Thornisde - Bang I Shot Him

The mel-spectogram seems to be less homogeneous and has more nuances in it, which should be more easily interpratable by neural networks. Therefore, we will be using mel-spectograms only to train our models in the rest of this paper.

Let's just point out the fact that we did not use colored spectograms but rather grey-scale spectogram since they are only coded on 1 channel (therefore dividing the number of features by 3).

4 CONVOLUTION NEURAL NETWORK

4.1 Architecture

After exprimenting different CNN architectures, we sticked with the following - rather simple - model, consiting of 3 convolution hidden layers, and applying max pooling to those layers :

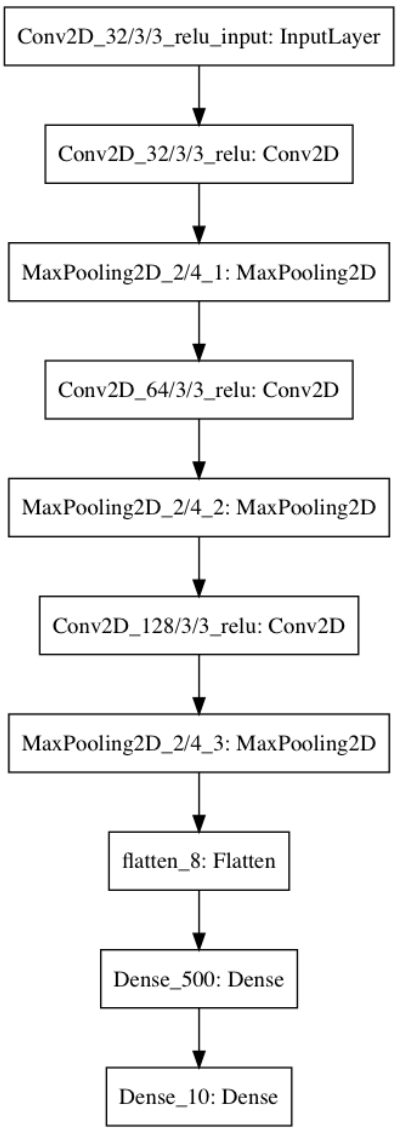


Figure 5 – Simple Convolution Neural Network

4.2 First results

The code to build the model is as follows :

```
import keras
from keras.models import Sequential
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.core import Flatten, Dense
from keras.optimizers iimport Adam
input_shape = (288,432,1)

model = Sequential()
model.add(Conv2D(32, 3, padding="same", input_shape=input_shape, activation="
    relu", name="Conv2D_32/3/3_relu"))
model.add(MaxPooling2D(pool_size=(2, 4), name="MaxPooling2D_2/4_1"))

model.add(Conv2D(64, 3, padding="same", activation="relu", name="Conv2D_64/3/3
    _relu"))
model.add(MaxPooling2D(pool_size=(2, 4), name="MaxPooling2D_2/4_2"))

model.add(Conv2D(128, 3, padding="same", activation="relu", name="Conv2D_128/3/3
    _relu"))
model.add(MaxPooling2D(pool_size=(2, 4), name="MaxPooling2D_2/4_3"))

model.add(Flatten())
model.add(Dense(500, activation="relu", name="Dense_500"))

model.add(Dense(10, activation="softmax", name="Dense_10"))

model.compile(loss="categorical_crossentropy", optimizer=Adam(), metrics=["
    accuracy"])
```

Runing an epoch took about 12 minutes, and we ran the model on 100 epochs, after which we obtained the following results :

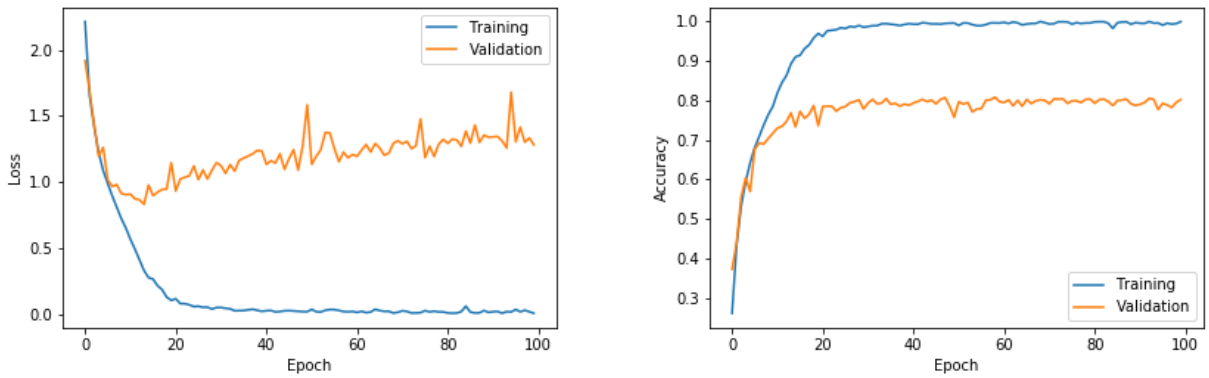


Figure 6 – Accuracy and loss plot fot the CNN

Through the whole project, we ran the neural network, using 85% of the data set in training and 15% a validation.

The results are rather satisfying, especially considering that state of the art is an accuracy of 90.79% [6].

However, the model seems to be clearly overfitting, and stops improving after 20 epochs (at which point the validation loss actually starts getting bigger). We tried to reduce that overfitting by introducing dropout, L2 regularization and batch normalization with different combinations, but this did not significantly diminish the overfitting.

4.3 Reducing mel-spectograms' resolution

We thought it could be intersting to reduce the resolution of our mel-spectograms, in order to train the network faster, and see what would be the tradeoff for accuracy.

4.3.1 • Resolution devided by 4

Instead of having images of size (432,228), we devided the number of pixels by two in each direction, thus resulting in images of size (216,144).
The epochs obviously ran faster (about 2 minutes per epoch), and the results were surpisingly good.

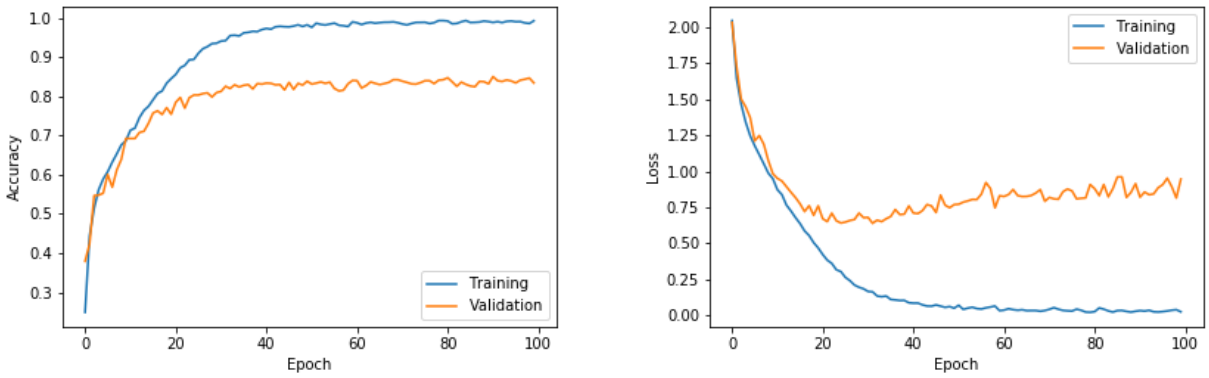


Figure 7 – Accuracy and loss plot fot the CNN with resolution devided by 4

4.3.2 • Resolution devided by 16

Instead of having images of size (432,228), we devided the number of pixels by four in each direction, thus resulting in images of size (108,72).

The epochs obviously ran way faster (a little less than 1 minute), and the results were still surpisingly good.

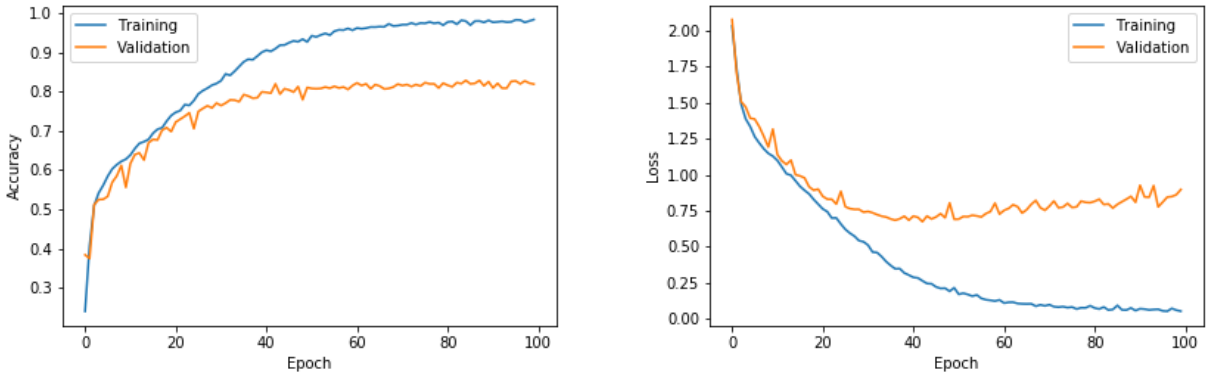


Figure 8 – Accuracy and loss plot fot the CNN with resolution devided by 16

4.3.3 • Insights

Suprinsingly enough, lowering the resolution of the data allows for a slightly better accuracy. Actually, this is most likely due to the fact that our CNN is not deep enough. Indeed since it doesn't have enough parameters and depth, the CNN cannot interpret well enough the huge amount of inputed features (124416 features in full resolution for images of size (432,288), coded on 1 channel). However, with our limited ressources, we could not have built deeper CNNs since it would take too much time to train.

It would have been extremely interesting to lower the resolution of our data when we first built our CNN, for it would have saved us a lot of computation time in trial and error.

Of the 3 models, the most accurate is clearly the one using data with a resolution devided by 16. The other two models have a similar confusion matrix and close validation accuracies.

5 LONG SHORT TERM MEMORY NEURAL NETWORK

The main difference between a Mel spectrogram and a clasic picture is that the Mel spectrogram has to be read from the left to the right. Indeed, the horizontal axis is temporal and the vertical is about frequency. Thus we had the idea to use Long Short Term Memory (LSTM) network, which is particularly performant with temporal series.

5.1 First Approach

Here is the architecture use for our LSTM network :

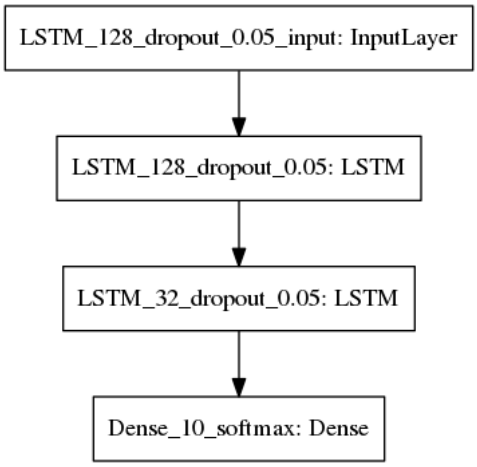


Figure 9 – LSTM Neural Network

Here's the Python code used to compute this neural network.

```
import keras
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers Adam
from keras.layers.recurrent import LSTM

input_shape = (288,432)

model = Sequential()

model.add(LSTM(units=128, dropout=0.05, recurrent_dropout=0.35, return_sequences
    =True, input_shape=input_shape))
```

```
model.add(LSTM(units=32, dropout=0.05, recurrent_dropout=0.35, return_sequences=
    False))

model.add(Dense(units=10, activation='softmax'))

model.compile(loss="categorical_crossentropy", optimizer=Adam(), metrics=["
    accuracy"])
```

After runing the model on 500 epochs of about 1 minute, we obtained the following results :

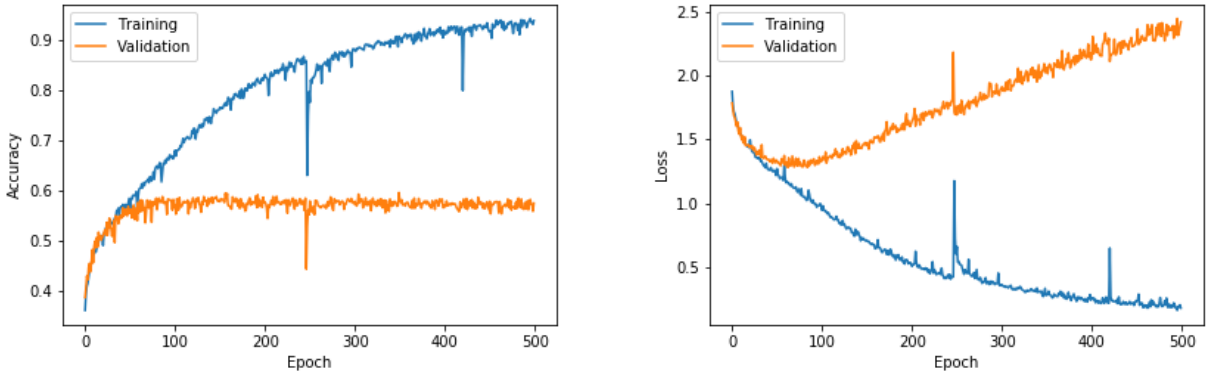


Figure 10 – Accuracy and loss plot fot the LSTM neural network

The model is strongly overfitting, after barely 50 epochs, and has lower perfomances than the previous CNN model. However, it runs way faster and has an extremely simple architecture (only 1 hidden layer !). Let's see if we can reduce the overfitting on this model.

5.2 Reducing Overfitting

The model previous model had some big overfitting issues, so we added and L2 regularizer, as well as more dropout :

```
import keras
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers Adam
from keras.layers.recurrent import LSTM
from keras.regularizers import l2

l2_labmda = 0.0001
input_shape = (288,432)

model = Sequential()

model.add(LSTM(units=128, dropout=0.05, recurrent_dropout=0.35, return_sequences
    =True, input_shape=input_shape, kernel_regularizer=l2(l2_lambda)))
```

```
14
15 model.add(LSTM(units=32, dropout=0.05, recurrent_dropout=0.35, return_sequences=
       False, kernel_regularizer=l2(l2_lambda)))
16
17 model.add(Dense(units=10, activation='softmax'))
18
19 model.compile(loss="categorical_crossentropy", optimizer=Adam(), metrics=["
       accuracy"])
```

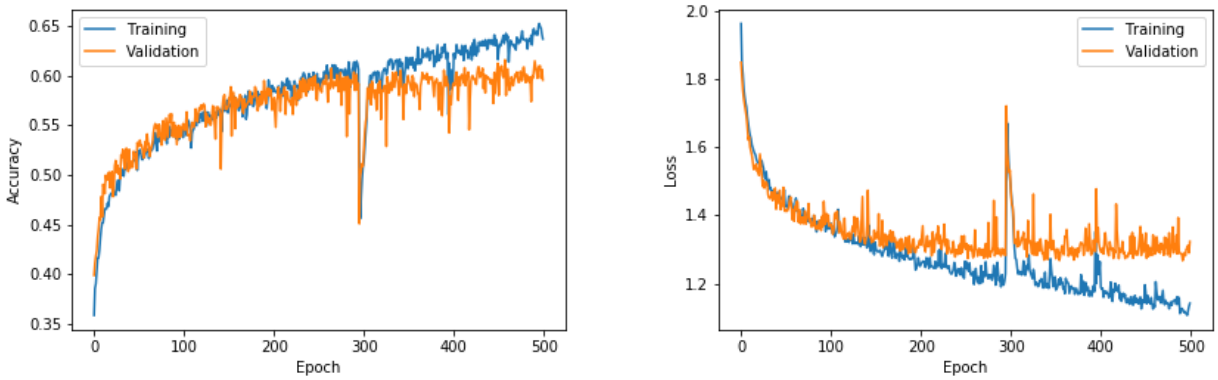


Figure 11 – Accuracy and loss plot for the improved LSTM neural network

Adding dropout and L2 regularization allowed to clearly improve the overfitting issues, even though there is still some room for improvement (there is still some overfitting, starting around epoch 300). The model eventualy achieves an accuracy of 60% after running for 500 epochs and could be more accurate if it ran on more epochs.

5.3 Reducing mel-spectograms' resolution

Seing as the CNN was producing quality results with reduced resolution we decided to test our improved LSTM neural network on the data with reduced resolution. Regarding the fact that this model has an extremely simple architecture, it is even more relevent to use data with a diminished resolution.

5.3.1 • Resolution devided by 4

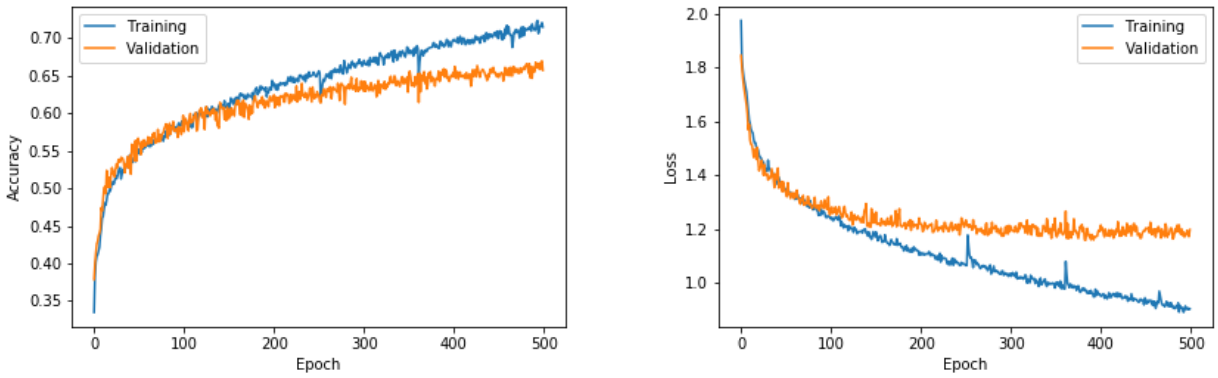


Figure 12 – Accuracy and loss plot fot the improved LSTM with resolution devided by 4

5.3.2 • Resolution devided 16

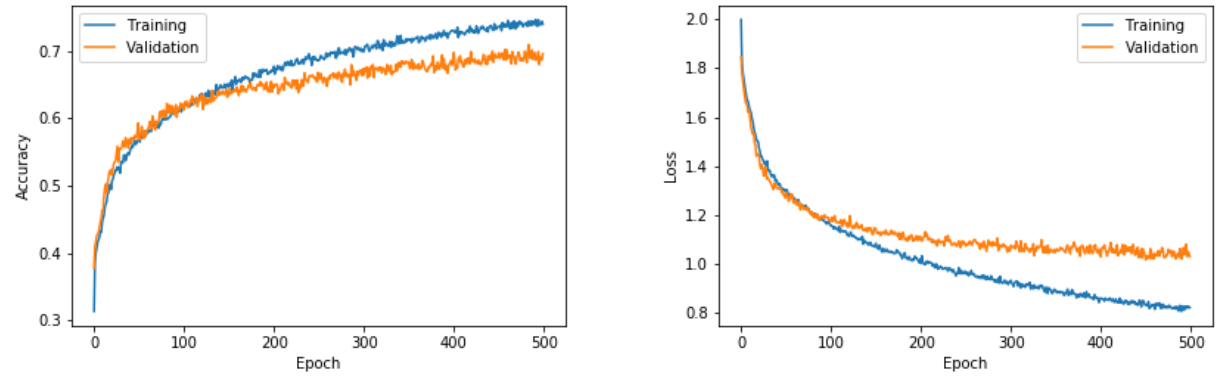


Figure 13 – Accuracy and loss plot fot the improved LSTM with resolution devided by 16

5.3.3 • Insights

For this simple model, the lower the resolution is, the better the prediction is. This shows the lack of depth and parameters in the model. If we had more computational power, we could have ran the models on more epochs to have better accuracies. We could also have added extra layers to ensure that the network can fully interpret the data.

6 STACKED MODELS

6.1 The idea

The two neural networks we used have different structures and thus aren't accurate for the same labels. Thus, it would be interesting to have a third model which takes the predictions from the two first models we used and give a more accurate one. The idea is that this third model will find on which label a model (CNN or LSTM) is accurate, and use strengths of a model to balance weaknesses of the other. This can be done by concatenating the outputs of the two first networks giving a (20,) vector. Then, we use this vector as an input for a third model. Here is the structure of the complete network :

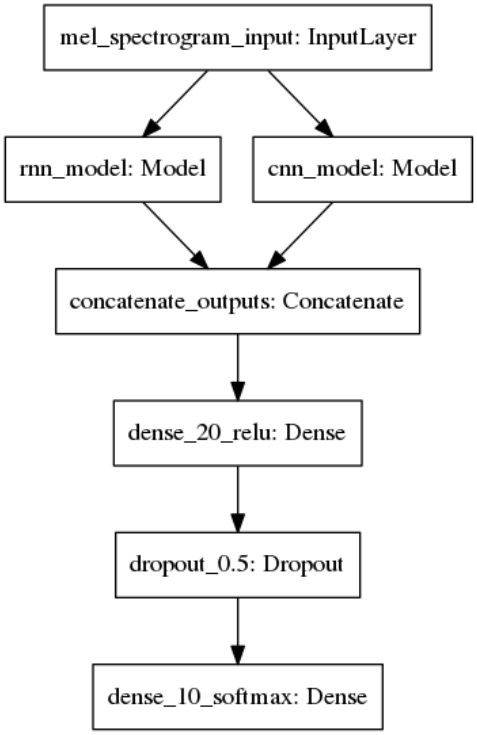


Figure 14 – LSTM and CNN combined

For the CNN and LSTM, the Sequential model (see [7]) of keras was enough, but as our new network is more complex, we had to use the Functional API (see [8]) which allows more sophisticated architectures of networks. Here is the Python code used to build this network :

```
input_shape = (288,432)

# we load models described in the previous parts, already trained
cnn_model = load_model('models/CNN_model.h5')
rnn_model = load_model('models/RNN_model.h5')

inputs = Input(shape=input_shape)

cnn_input = Reshape(input_shape+(1,))(inputs)
rnn_input = inputs

rnn_output = rnn_model(rnn_input)
cnn_output = cnn_model(cnn_input)

rnn_model = Model(inputs, rnn_output, name='rnn_model')
# the next loop freezes the weights of the RNN model
for layer in rnn_model.layers:
    layer.trainable = False

cnn_model = Model(inputs, cnn_output, name='cnn_model')
# the next loop freezes the weights of the CNN model
for layer in cnn_model.layers:
    layer.trainable = False

x = keras.layers.concatenate([rnn_model(inputs), cnn_model(inputs)], name='
    concatenate_outputs')
x = Dense(20, activation='relu', name='dense_20_relu')(x)
x = Dropout(0.5, name='dropout_0.5')(x)
output = Dense(10, activation='softmax', name='dense_10_softmax')(x)

model = Model(inputs=inputs, outputs=output)
model.compile(loss="categorical_crossentropy", optimizer=Adam(), metrics=["
    accuracy"])
```

We decided to freeze the weights of the CNN and RNN because we considered that they were already enough trained and we wanted to train only the weights of the last dense layers.

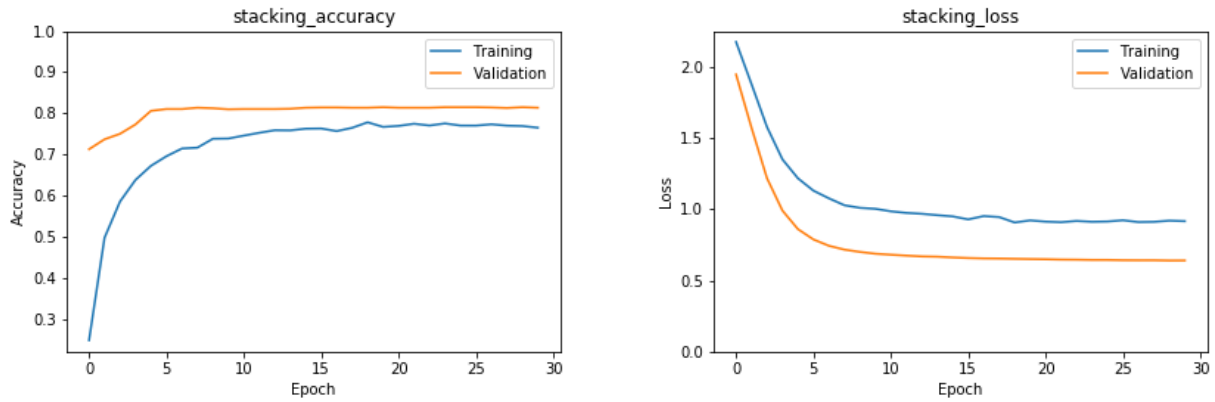


Figure 15 – Accuracy and loss plot fot the stacked model (full resolution)

We can see the convergence is very fast (about 5 epochs for the accuracy on test set). This is normal since we added two "simple" layers". But the improvement in the accuracy on the test set is really weak compared to the one achieved with the CNN.

The major improvement of this model lies in achieving to completely get rid of overfitting, whilst other models had significant overfitting issues.

6.2 Reducing mel-spectograms' resolution

As both models improved their accuracy by reducing the size of the input spectrograms, we decided to test the stacked models for those resolutions. The results are satisfying, for we achieved more than 85% accuracy with the stacked model, with a resolution divided by 16.

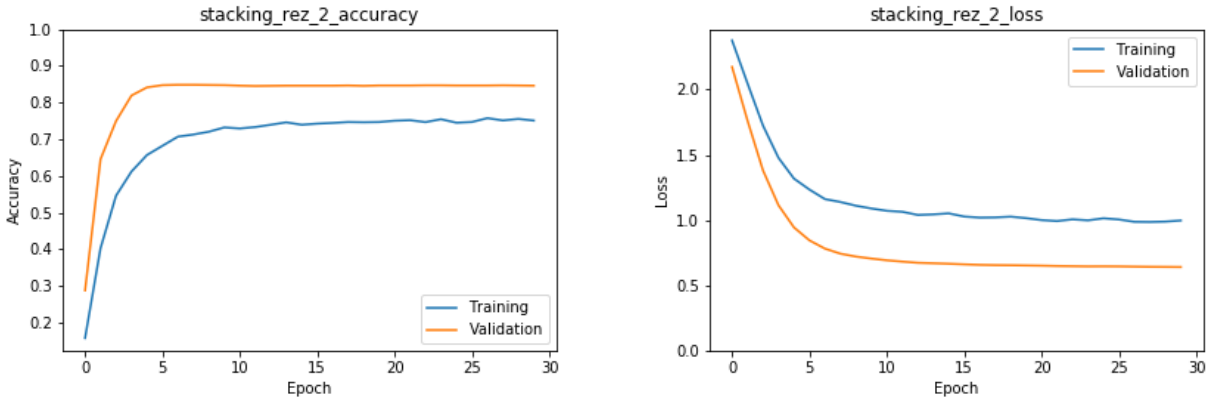


Figure 16 – Accuracy and loss plot fot the stack model with resolution devided by 4

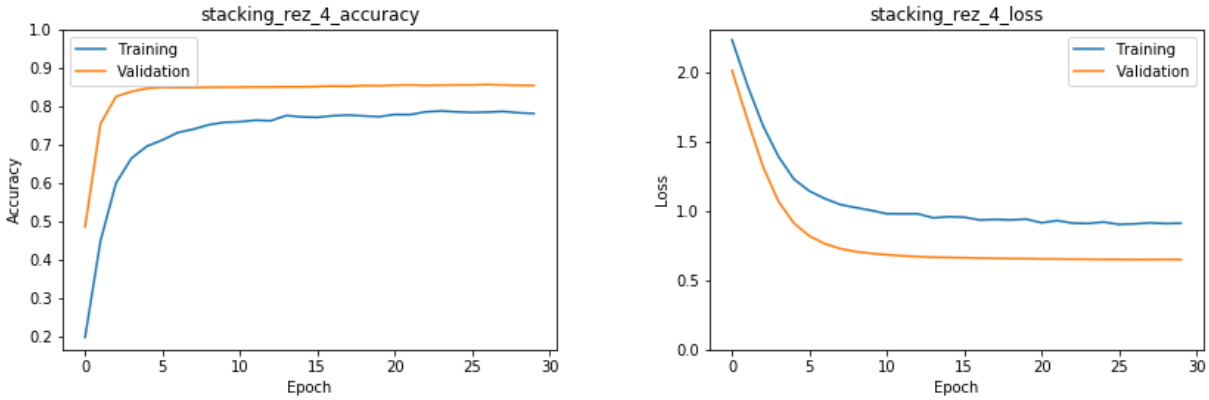


Figure 17 – Accuracy and loss plot fot the stack model with resolution devided by 16

7 RESULTS COMPARISON AND POSSIBLE IMPROVEMENTS

Table 1 – Comparaison of the models' performances

Model	Resolution	Epochs	Min/Epoch	Overfitting	**Acc.**
CNN	Full	100	12	Huge	**80%**
CNN	Divided by 4	100	1.5	Huge	**84%**
CNN	Divided by 16	100	0.5	Huge	**82%**
LSTM	Full	500	1	Small	**60%**
LSTM	Divided by 4	500	0.75	Small	**67%**
LSTM	Divided by 16	500	0.15	Small	**69%**
CNN+LSTM	Full	30	4	None	**81.5%**
CNN+LSTM	Divided by 4	30	1	None	**84.5%**
CNN+LSTM	Divided by 16	30	0.25	None	**85.3%**

The results we achieved so far are pretty satisfying, and we're convinced that our models could be improved on several issues.

First of all, we only used Mel's spectrograms, but we could also imagine neural networks taking classic spectrograms as input, and stack those models to improve the accuracy. Indeed, those different spectrograms probably don't highlight same caracteristics of a song, and are certainly very complementary.

Moreover, had we more computing power we would have tried to use more data from the Magnatagatune dataset (we had an initial database of about 25.000 song extracts, but we only managed to use around 10.000 of them, because we had difficulty to map the songs we had to their music genres). More time spent on this part would be interesting for the project. Indeed, more inputed data would have allowed us to build deeper models, which would have definitly proven to be more accurate.

There are many more music genres than the one we used, and it would be interesting to have a more precise classification of each song. Thus we could imagine predicting the general music genre (Rock, Classic, Electro...) and then, in regards of this first prediction, predict the subgenre of this song.

A
CONFUSION MATRICES

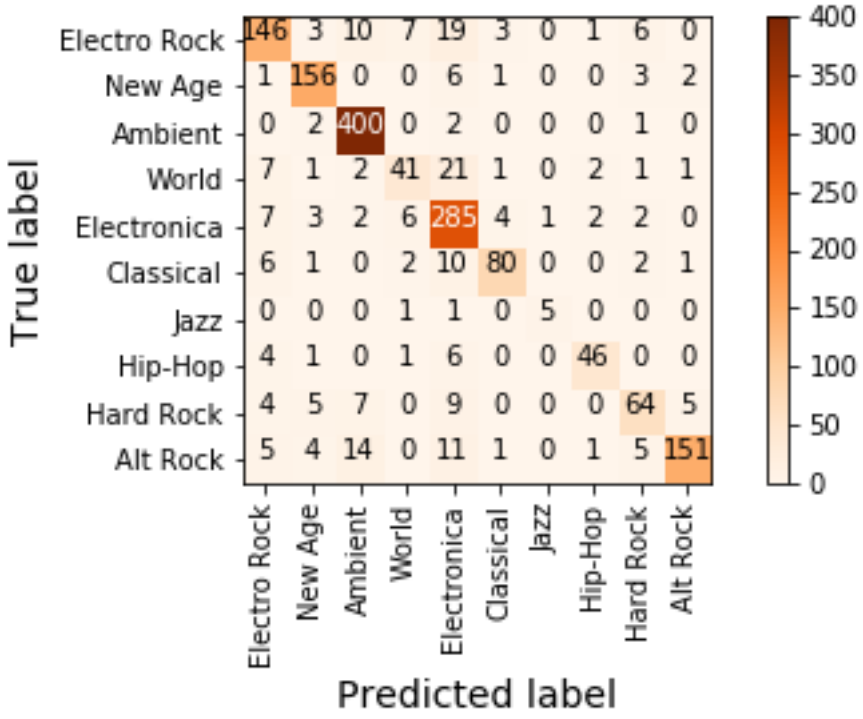


Figure A.1 – Confusion Matrix for the CNN

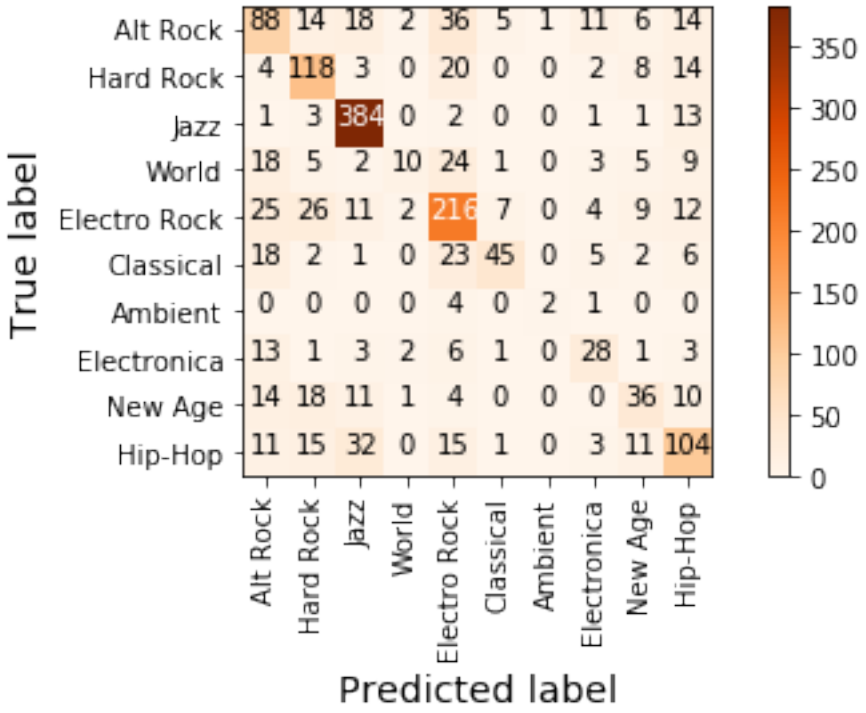


Figure A.2 – Confusion Matrix of the improved LSTM neural network

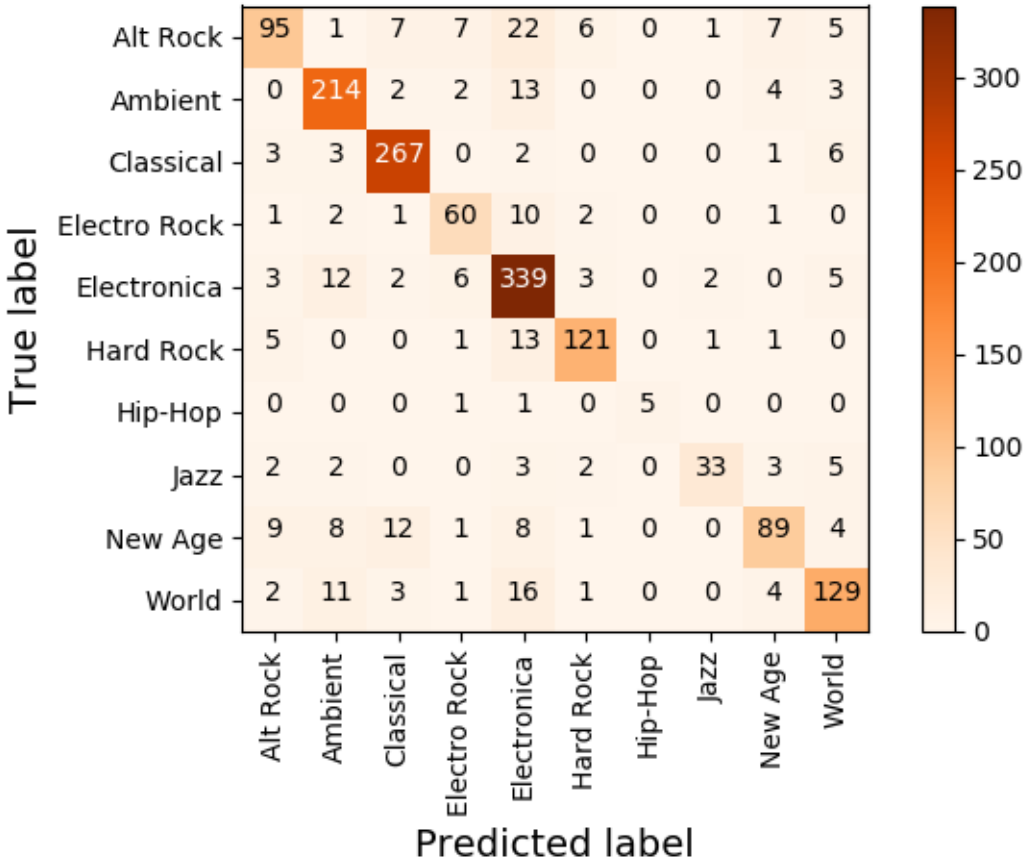


Figure A.3 – Confusion Matrix of the stack model with original resolution

BIBLIOGRAPHY

[1] City University London. The magnatune dataset. http ://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset.

[2] Yann Le Cun. Mnist dataset. http ://yann.lecun.com/exdb/mnist/.

[3] Bartosz Michalak Piotr Kozakowski. Music genre recognition. http ://deepsound.io/music_genre_recognition.html.

[4] Daniel Grzywczak Grzegorz Gwardys. Deep image features in music information retrieval. https ://www.degruyter.com/downloadpdf/j/eletel.2014.60.issue-4/eletel-2014-0042/eletel-2014-0042.pdf.

[5] Tao Feng. Deep learning for music genre classification. https ://-courses.engr.illinois.edu/ece544na/fa2014/Tao_Feng.pdf.

[6] Guojun Lu Arash Foroughmand Arabi. Enhanced polyphonic music genre classification using high level features. http ://ieeexplore.ieee.org/stamp/stamp.jsp ?tp=&arnumber=5478635.

[7] François Chollet et al. Keras sequential model. https ://keras.io/getting-started/sequential-model-guide/, 2015.

[8] François Chollet et al. Keras functional api. https ://keras.io/getting-started/functional-api-guide/, 2015.